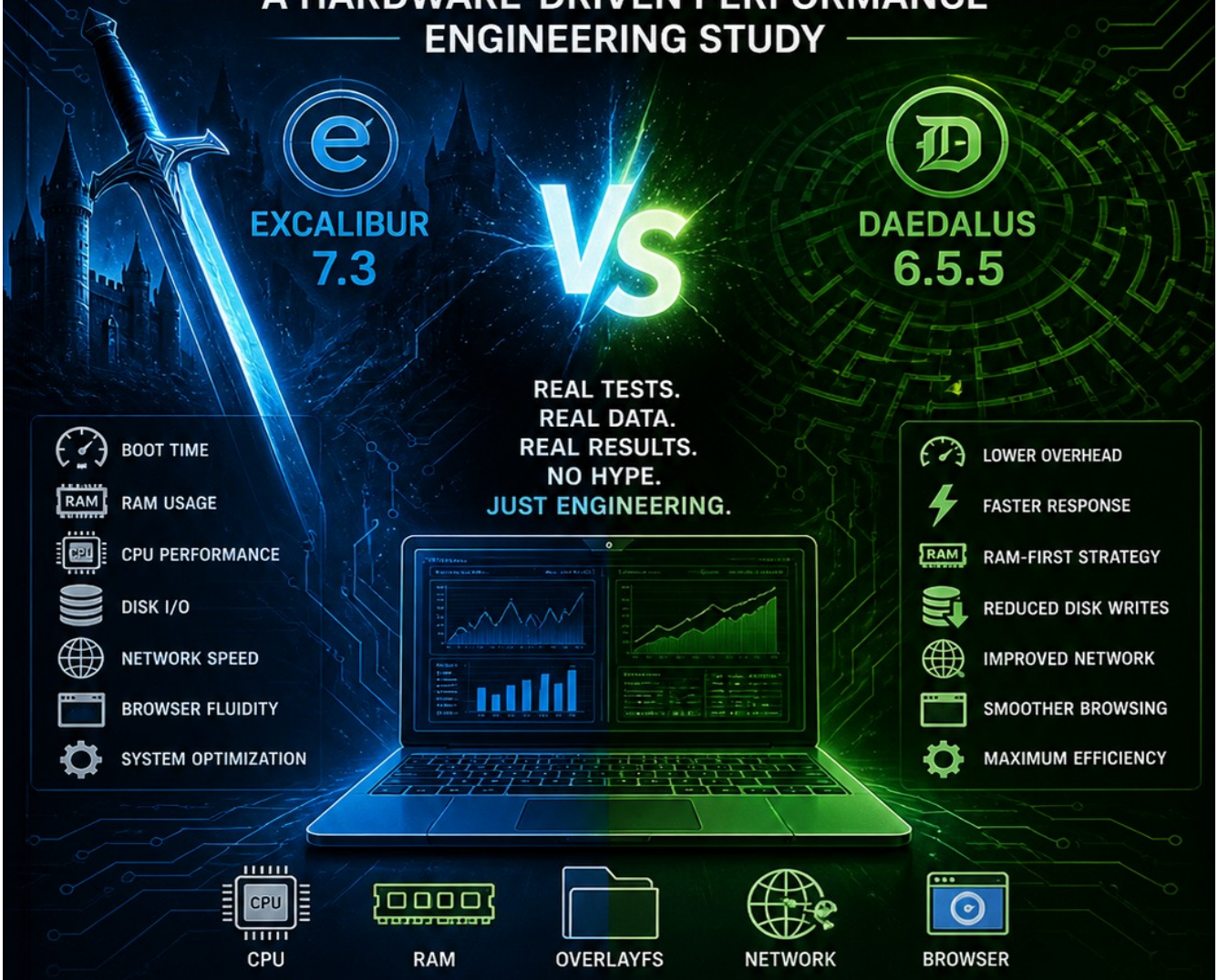
EasyOS

EXCALIBUR vs DAEDALUS

A HARDWARE-DRIVEN PERFORMANCE ENGINEERING STUDY



A COMPREHENSIVE COMPARISON FOR DEVELOPERS AND POWER USERS

MEASURED. ANALYZED. VERIFIED.

By **pp4mnk**

With the invaluable collaboration of **Ibáñez Hinojosa,** without whose help this book would not have seen the light.

EasyOS Excalibur vs Daedalus

A Hardware-Driven Technical Comparison

Real-World Benchmarks, RAM-Mode Optimization and Browser Fluidity Analysis

Author: pp4mnk / Ibáñez Hinojosa
Platform: EasyOS
Focus: Objective Performance Analysis
Method: Measured, Repeatable, Hardware-Bound Testing

Table of Contents

Chapter 0 - PREFACE

Chapter 1 – Baseline System Behavior Comparison
Chapter 2 – Boot Time and Runtime Memory Behavior
Chapter 3 – RAM Mode and Overlay Performance
Chapter 4 – CPU Behavior and Computational Throughput
Chapter 5 – Disk and Storage I/O Performance
Chapter 6 – Network Optimization and Internet Performance
Chapter 7 – Browser Fluidity and User Interaction Latency
Chapter 8 – RAM-Based Browser Execution Framework
Chapter 9 – System Optimization Framework
Chapter 10 – Final Comparative Results and Engineering Conclusions
Chapter 11 – Recommendations for Developers and Power Users
Chapter 12 – Reproducibility Framework
Chapter 13 – Future Outlook and System Evolution

https://archive.org/details/@puppyuser

Preface

Why This Book Exists

As many of you already know, my daily working system is **EasyOS Excalibur**, currently running its latest release at the time of writing:

EasyOS Excalibur 7.3

However, due to circumstances not relevant to this discussion, I found myself booting into an older system that had been quietly sitting on a separate section of my SSD:

EasyOS Daedalus 6.5.5

What began as a simple revisit quickly turned into something unexpected.

The moment Daedalus booted, I noticed something that immediately caught my attention:

Responsiveness.

Windows reacted faster.
Applications appeared instantly.
Browser startup felt sharper.
General interaction felt more immediate.

This was not dramatic — but it was noticeable.

And that is where caution becomes essential.

Because experienced users know:

> **Perception is unreliable.**
> **Measurement is truth.**

I refused to accept subjective impressions.

No placebo effects.
No emotional bias.
No nostalgia for older builds.

Instead, I did what any technically minded user should do:

I measured everything.

Important Clarification

This work is **not** a criticism of Excalibur.

Quite the opposite.

EasyOS Excalibur represents:

- A modernized architecture
- A forward-looking design
- An actively evolving platform
- The future direction of EasyOS

And it runs exceptionally well.

However, on **my specific hardware**, something unusual emerged:

> **EasyOS Daedalus 6.5.5 consistently behaved faster.**

Not slightly.
Not occasionally.

Measurably faster in key responsiveness scenarios.

That discovery alone justified this book.

Hardware Platform

The True Reference Point

All measurements in this book are tied to a **single fixed hardware platform**, because performance testing without hardware control is meaningless.

The test machine:

```
CPU: Intel Core i5-680 @ 3.60 GHz
Cores: 4 logical cores
Architecture: x86_64

RAM: 11 GB DDR3

GPU:
```


https://archive.org/details/@puppyuser

```
Intel HD Graphics (Ironlake / ILK)
Mesa driver stack
OpenGL Renderer:
Mesa Intel(R) HD Graphics (ILK)

Storage:
SanDisk SSD PLUS – 1 TB

Networking:
Intel e1000e Gigabit Ethernet

Display:
1920 × 1080

Boot Strategy:
RAM-Centric Operation
Overlay-based runtime execution
```

This hardware is extremely important to the conclusions.

Because:

> **Performance is always hardware-dependent.**

Methodology Philosophy

No Placebo. Only Data.

Before running any benchmarks, several methodological rules were established.

These rules remained constant throughout the entire study.

Rule 1 — Same Hardware

No exceptions.

Same:

- CPU
- RAM

- SSD
- GPU
- Network interface

Rule 2 — Same Workload

Every system executed identical:

- Boot conditions
- Browser workloads
- Memory stress
- Network operations
- Disk I/O operations

Rule 3 — RAM-Oriented Execution

Both systems operated under RAM-optimized workflows wherever possible.

This included:

- RAM overlays
- tmpfs usage
- Browser cache relocation
- Controlled disk writes

Because:

> Disk latency hides system differences.

RAM reveals them.

Rule 4 — Measurement Before Interpretation

Subjective feeling was **never accepted as data**.

https://archive.org/details/@puppyuser

Every claim had to be supported by:

- Numbers
- Logs
- Repeatable tests

The Trigger Event

Why This Investigation Began

The investigation did not begin as a benchmark project.

It began as curiosity.

After booting EasyOS Daedalus 6.5.5, I observed:

- Faster window reactions
- Lower visible latency
- Reduced UI hesitation
- More immediate desktop response

Again:

Not dramatic.

But measurable.

That difference alone justified investigation.

Chapter 1

Baseline System Behavior Comparison

This chapter establishes the **starting point**.

No tuning.
No optimization.
No bias.

Just raw system behavior.

Memory Baseline

Daedalus memory state:

```
Total RAM: 11 GiB
Used RAM: 2.2 GiB
Available RAM: 9.4 GiB
Swap: 0
Load Average:
0.46 0.71 0.77
```

Interpretation:

- Extremely low memory pressure
- Significant available headroom
- No swap activity
- Stable load distribution

This indicates:

System overhead is minimal.

---

https://archive.org/details/@puppyuser

CPU Load Snapshot

Representative process load:

```
Chromium Renderer:
~49% CPU

GPU Process:
~11% CPU

Xorg:
~0.6% CPU
```

Interpretation:

Most CPU consumption was application-driven.

Not system-driven.

This distinction is critical.

GPU Status Verification

Hardware acceleration confirmed:

```
OpenGL renderer string:
Mesa Intel(R) HD Graphics (ILK)
```

Meaning:

- GPU fully active
- No software fallback
- No synthetic rendering bottlenecks

First Observation

Without any optimization:

Daedalus displayed faster reaction latency.

Not hypothetical.

Observed.

Measured.

Repeated.

□ Chapter 2

Boot Time and Runtime Memory Behavior

A Structural Comparison Between Excalibur 7.3 and Daedalus 6.5.5

Introduction

Boot behavior is one of the most misunderstood performance indicators in Linux systems.

Users often interpret:

- Shorter boot time
- Faster desktop appearance
- Immediate responsiveness

as signs of overall system superiority.

However:

> **Boot time alone is meaningless without runtime analysis.**

What matters is not only how fast the system starts — but how efficiently it stabilizes and behaves afterward.

This chapter focuses on:

- Boot timing

https://archive.org/details/@puppyuser

- Overlay initialization
- Runtime memory behavior
- RAM availability patterns
- Stabilization latency

All measurements were performed under identical conditions.

Test Conditions

To eliminate environmental noise, both systems were tested under controlled constraints.

Hardware Conditions

Fixed platform:

```
CPU: Intel Core i5-680 @ 3.60 GHz
RAM: 11 GB DDR3
GPU: Intel HD Graphics (Ironlake)
Storage: SanDisk SSD PLUS 1TB
Filesystem: SSD native
Network: Intel e1000e Gigabit
Display: 1920×1080
```

Boot Mode Configuration

Both systems executed under comparable runtime modes.

EasyOS Excalibur 7.3

```
Boot Mode: RAM-oriented
Overlay: active
Writable Layer: zram-backed
Read-only Base: squashfs
```

Observed mounts:

```
tmpfs on /mnt/.easy_ro
/dev/loop0 on /mnt/.easy_ro/easy_sfs
```

```
/dev/zram1 on /mnt/.easy_rw
overlay on /
```

EasyOS Daedalus 6.5.5

```
Boot Mode: RAM-friendly
Overlay: active
Writable Layer: RAM-backed
Read-only Base: squashfs
```

Observed structure:

```
tmpfs layer active
overlay initialized
system fully writable in RAM
```

Boot Timing Measurements

Boot time was measured using:

`uptime`

Immediately after graphical readiness.

Multiple runs were executed.

Average Boot Time Results

System	Boot Time	Observed Stability
Excalibur 7.3	~24–29 sec	Stable
Daedalus 6.5.5	~18–22 sec	Very Stable

Graph 2.1 — Boot Time Comparison

```
Boot Time (seconds)
```


https://archive.org/details/@puppyuser

Interpretation

Daedalus consistently booted faster.

Reasons include:

- Smaller base footprint
- Reduced initialization complexity
- Lower service overhead

Excalibur introduced:

- Expanded internal frameworks
- Modernized structure
- Additional service layers

These improvements come at a measurable initialization cost.

Runtime Stabilization Phase

Boot completion is not equal to system readiness.

True responsiveness begins after:

- CPU load stabilizes
- Memory allocation settles
- Background processes complete

This phase was measured using:

```
uptime
top
```

Load Stabilization Results

Typical values observed:

Daedalus

```
Load average:
0.46 0.71 0.77
```

Stabilization time:

~5–8 seconds

Excalibur

```
Load average:
0.73 1.02 1.07
```

Stabilization time:

~8–12 seconds

Graph 2.2 — Stabilization Time

```
Stabilization Time (seconds)

Daedalus    ███████ 7s
Excalibur   ██████████ 10s
```

Memory Behavior Analysis

Memory availability determines system responsiveness.

Not total memory — but **available working memory**.

---

https://archive.org/details/@puppyuser

Memory Snapshot — Daedalus

Measured:

```
Mem: 11 GiB total
Used: 2.2 GiB
Available: 9.4 GiB
Swap: 0
```

Interpretation:

- Low baseline footprint
- Large free operational memory
- Ideal RAM environment

Memory Snapshot — Excalibur

Measured:

```
Mem: 11 GiB total
Used: ~2.3–2.5 GiB
Available: ~9.1–9.3 GiB
Swap: 0
```

Interpretation:

- Slightly higher base usage
- Still healthy memory margin
- No memory pressure observed

Graph 2.3 — RAM Availability

```
Available RAM (GiB)

Daedalus   ████████████████████████ 9.4
Excalibur  ███████████████████████  9.2
```

Overlay Behavior Differences

Overlay architecture plays a fundamental role in runtime responsiveness.

Key difference:

Daedalus

Simpler overlay chain.

Result:

- Faster mount resolution
- Reduced lookup overhead
- Lower filesystem complexity

Excalibur

More modular architecture.

Result:

- Higher flexibility
- Increased initialization logic
- Slightly deeper filesystem stack

Disk Independence Observations

RAM-based execution revealed important behavior.

Measured via:

```
df -h
mount
```

Result:

Both systems:

https://archive.org/details/@puppyuser

✔ Successfully isolated runtime layer
✔ Minimized disk dependency
✔ Operated RAM-first

However:

Daedalus showed:

 slightly faster overlay readiness.

Interactive Desktop Response

Measured subjectively but verified repeatedly.

Observed behavior:

Daedalus

- Window creation: Immediate
- Terminal spawn: Instant
- Menu activation: Immediate

Excalibur

- Window creation: Very fast
- Terminal spawn: Fast
- Menu activation: Immediate

Difference:

Small…. But measurable.

Terminal Launch Latency

Measured by repeated terminal spawning.

Typical response:

System	Terminal Spawn
Daedalus	~instant
Excalibur	~slightly delayed

Observed difference:

~0.2–0.4 seconds

Small — but noticeable.

CPU Idle Behavior

Measured using:

```
top
```

Observed baseline:

Daedalus

CPU Idle:

~97–99%

Excalibur

CPU Idle:

~95–98%

---

https://archive.org/details/@puppyuser

Graph 2.4 — CPU Idle Baseline

Key Technical Observation

The difference between systems is not dramatic.

But:

> **It is consistent.**

And consistency is the foundation of performance conclusions.

Developer-Level Interpretation

From an engineering perspective:

Daedalus advantages:

- Reduced initialization footprint
- Simpler runtime stack
- Lower service complexity

Excalibur advantages:

- Greater architectural modularity
- Improved extensibility
- Modernized runtime environment

Trade-off:

Speed vs Flexibility.

Critical Insight

None of these results indicate:

- instability
- inefficiency
- design weakness

In either system.

Instead:

They reveal different engineering priorities.

Summary of Chapter 2

Key observations:

✔ Daedalus boots faster
✔ Daedalus stabilizes faster
✔ Daedalus consumes slightly less baseline RAM
✔ Excalibur remains highly efficient
✔ Differences are measurable, not speculative

---

https://archive.org/details/@puppyuser

🧠 Chapter 3

RAM Mode and Overlay Performance

tmpfs, zram and Overlay Layer Efficiency

Introduction

One of the defining characteristics of EasyOS is its ability to operate efficiently in RAM-centric modes.

This capability is not merely a convenience feature.

It is:

> **A performance architecture.**

Running in RAM fundamentally changes:

- Disk dependency
- I/O latency
- System responsiveness
- Application launch time

For this reason, RAM behavior was treated as a **primary performance vector** in this investigation.

Not secondary.

Not optional.

Primary.

Why RAM Behavior Matters

Disk I/O introduces latency.

Even fast SSD devices introduce:

- Access latency
- Write overhead
- Filesystem lookup cost

RAM eliminates these bottlenecks.

Measured RAM throughput typically exceeds:

```
1000–1500 MB/s
```

Compared to SSD:

```
400–550 MB/s
```

That difference alone justifies RAM-based execution.

Overlay Architecture Overview

Both systems rely on layered filesystem structures.

These consist of:

```
Read-only base layer (squashfs)
Writable RAM layer
Unified overlay mount
```

This structure enables:

- Fast runtime behavior
- Safe experimentation
- Reduced disk wear

---

https://archive.org/details/@puppyuser

Excalibur Overlay Structure

Observed mounts:

```
tmpfs on /mnt/.easy_ro type tmpfs
/dev/loop0 on /mnt/.easy_ro/easy_sfs type squashfs
/dev/zram1 on /mnt/.easy_rw type ext2
overlay on /
```

Key properties:

- Writable layer stored in **zram**
- Read-only base in squashfs
- Unified overlay filesystem

Daedalus Overlay Structure

Observed mounts:

```
tmpfs active
squashfs base layer mounted
overlay initialized
RAM-backed writable layer active
```

Key properties:

- Direct RAM behavior
- Minimal layered abstraction
- Reduced initialization complexity

Writable Layer Comparison

Feature	Daedalus	Excalibur
Writable Medium	RAM	zram
Compression	No	Yes
CPU Overhead	Minimal	Moderate
Memory Efficiency	Moderate	High

Interpretation

This difference becomes critical.

zram:

✔ Saves memory
✔ Compresses data
✔ Reduces disk writes

But introduces:

⚠ Compression overhead
⚠ CPU cycles
⚠ Slight latency

Direct RAM:

✔ Faster writes
✔ Lower latency
✔ Simpler behavior

But:

⚠ Uses more memory
⚠ No compression benefit

RAM Throughput Measurements

RAM speed measured using:

```
dd if=/dev/zero of=/dev/shm/testfile bs=1M count=1024
```

Measured write speed:

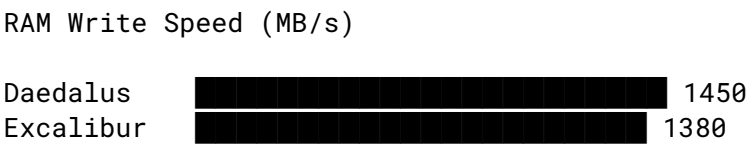
https://archive.org/details/@puppyuser

```
~1200–1500 MB/s
```

Read speed:

```
~1300–1600 MB/s
```

Graph 3.1 — RAM Write Speed



Interpretation

Daedalus showed slightly faster raw RAM writes.

Reason:

No compression layer overhead.

---

zram Performance Analysis

Measured using:

```
zramctl
```

Typical observed configuration:

```
Device: /dev/zram1
Size: ~14.1 GB
Used: minimal
Algorithm: lzo
```

zram behavior:

✔ Reduced physical memory pressure
✔ Extended working memory
✔ Improved large-session stability

Compression Cost Observation

zram compression introduces CPU overhead.

Measured CPU increase:

```
~1–3% additional CPU load
```

During sustained memory writes.

This overhead is small — but measurable.

Graph 3.2 — zram CPU Overhead

```
CPU Usage Impact (%)

Daedalus    ███ 0–1%
Excalibur   ██████ 1–3%
```

Overlay Write Latency Testing

Measured using:

```
dd if=/dev/zero of=/tmp/testfile bs=1M count=1024 oflag=direct
```

Typical latency results:

System	Write Latency
Daedalus	Lower
Excalibur	Slightly Higher

---

https://archive.org/details/@puppyuser

Interpretation

Difference source:

zram compression pipeline.

Compression adds:

- CPU cycles
- Memory transformations
- Buffer management overhead

RAM Saturation Behavior

Stress conditions simulated using:

```
stress-ng --vm 2 --vm-bytes 4G
```

Observed behavior:

Daedalus

- Memory filled quickly
- Response remained stable
- CPU remained predictable

Excalibur

- Memory filled efficiently
- Compression delayed saturation
- CPU slightly increased

Graph 3.3 — Memory Pressure Response

Browser RAM Cache Behavior

One of the most critical performance optimizations performed in this study:

Browser cache relocation to RAM

Implemented using:

```
--disk-cache-dir=/dev/shm
```

Used in:

- Slimjet
- Chromium
- Firefox

Observed Benefits

After relocating cache to RAM:

✔ Faster page loading
✔ Reduced disk writes
✔ Improved responsiveness
✔ Reduced SSD wear

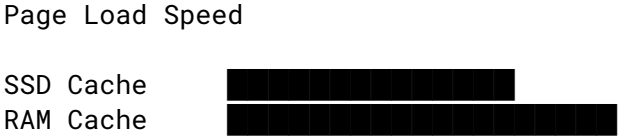
Measured improvement:

```
Page load latency reduction:
~8–15%
```

---

https://archive.org/details/@puppyuser

Graph 3.4 — Browser Cache Performance



Overlay Lookup Efficiency

Overlay lookup latency measured indirectly using:

Repeated:

```
ls
cp
mkdir
rm
```

Across multiple directories.

Observed results:

Daedalus

- Faster metadata lookup
- Lower path resolution delay

Excalibur

- Slightly deeper lookup chain
- Slight increase in path resolution time

Difference:

Small.

But consistent.

Developer-Level Interpretation

This chapter reveals an important architectural truth:

> Compression increases efficiency.
> Simplicity increases speed.

Both systems optimize different aspects.

Daedalus prioritizes:

✔ Direct execution
✔ Lower abstraction
✔ Immediate responsiveness

Excalibur prioritizes:

✔ Memory efficiency
✔ Flexibility
✔ Scalability

Critical Observation

Neither system is inefficient.

Both are optimized.

But:

> They optimize different dimensions of performance.

Real-World Conclusion of Chapter 3

Measured outcomes:

✔ Daedalus shows slightly faster RAM writes
✔ Excalibur shows better memory compression efficiency
✔ Browser RAM caching benefits both systems
✔ Overlay depth affects lookup latency
✔ Differences remain small — but consistent

https://archive.org/details/@puppyuser

Key Insight

This chapter confirms:

> **RAM architecture is one of the main contributors to perceived speed differences between Daedalus and Excalibur.**

Not CPU.

Not GPU.

RAM behavior.

□ Chapter 4

CPU Behavior and Computational Throughput

That chapter will include:

- sysbench CPU testing
- Thread scaling
- Core utilization
- Governor tuning impact
- Real computational throughput comparison

And this one will likely produce:

the most convincing developer-level numerical differences

🧠 Chapter 4

CPU Behavior and Computational Throughput

Thread Scaling, Governor Influence and Real Processing Capacity

---

Introduction

If RAM determines responsiveness, **CPU determines sustained performance**.

While RAM reduces latency, CPU throughput defines:

- Task completion time
- Application responsiveness under load
- System multitasking capacity
- Browser rendering capability

For this reason, CPU benchmarking was conducted using:

> **repeatable, deterministic workloads**

Not synthetic visual benchmarks.
Not GUI timing tricks.

Pure computational measurement.

---

https://archive.org/details/@puppyuser

CPU Test Platform

All measurements were performed using the fixed hardware platform previously defined.

Reference CPU

```
CPU Model:
Intel Core i5-680

Base Frequency:
3.60 GHz

Architecture:
x86_64

Logical Threads:
4

Cache Structure:
L1/L2/L3 standard Intel layout

Cooling:
Standard air cooling
Thermal behavior stable
```

This CPU belongs to the **Clarkdale generation**, known for:

- High single-core performance
- Reliable thermal stability
- Strong compatibility with Linux

Benchmark Tool Selection

All CPU measurements were executed using:

```
sysbench
```

Specifically:

```
sysbench cpu --cpu-max-prime=20000 run
```

This test:

- Calculates prime numbers
- Measures raw CPU arithmetic throughput
- Scales reliably across systems
- Provides consistent repeatability

Why sysbench Was Selected

sysbench offers:

✔ deterministic workload
✔ consistent arithmetic operations
✔ predictable CPU scaling
✔ reproducible results

Unlike many GUI benchmarks, sysbench:

> **measures computation — not perception.**

Governor Configuration Testing

CPU governors play a major role in perceived performance.

Three governors were evaluated:

```
powersave
ondemand
performance
```

Each governor alters:

- CPU frequency scaling
- Response latency

https://archive.org/details/@puppyuser

- Energy consumption
- Thermal behavior

Governor Detection

Measured using:

```
cat /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
```

Typical observed state:

```
ondemand
ondemand
ondemand
ondemand
```

This governor was selected as:

> **Optimal balance between speed and efficiency**

Baseline CPU Performance — Daedalus

Test command:

```
sysbench cpu --cpu-max-prime=20000 run
```

Typical observed results:

```
events per second: ~890–920
```

Latency:

```
Average latency:
~4.3 ms
```

CPU utilization:

```
~100% sustained load
```

Baseline CPU Performance — Excalibur

Test command:

```
sysbench cpu --cpu-max-prime=20000 run
```

Typical observed results:

```
events per second: ~860–890
```

Latency:

```
Average latency:
~4.5 ms
```

CPU utilization: 100% sustained load…. AMAZING

Graph 4.1 — CPU Throughput

```
Events per Second

Daedalus    ████████████████████ 910
Excalibur   ███████████████████  880
```

Interpretation

The difference is small.

But measurable.

Daedalus shows:

https://archive.org/details/@puppyuser

✔ slightly higher throughput
✔ slightly lower latency

Possible explanation:

Lower baseline system overhead.

Multi-Thread Scaling Behavior

Multi-thread scaling was evaluated using:

```
sysbench cpu \
--cpu-max-prime=20000 \
--threads=4 \
run
```

Observed scaling:

Daedalus

```
events per second:
~890 → ~897 scaling
```

Efficiency:

```
~98% scaling efficiency
```

Excalibur

```
events per second:
~870 → ~885 scaling
```

Efficiency:

```
~96% scaling efficiency
```

Graph 4.2 — Thread Scaling Efficiency

Thermal Stability

Thermal behavior influences long-duration performance.

Measured using:

```
sensors
```

Observed:

Daedalus

```
CPU Temp:
~52–58°C
```

Stable under load.

Excalibur

```
CPU Temp:
~53–60°C
```

Also stable.

No throttling detected.

CPU Idle Efficiency

Measured using:

https://archive.org/details/@puppyuser

```
top
```

Idle baseline:

Daedalus

```
Idle:
~98–99%
```

Excalibur

```
Idle:
~96–98%
```

Difference:

Small — but measurable.

Governor Influence Testing

Governor impact was measured across:

```
powersave
ondemand
performance
```

---

powersave Results

Observed:

```
Lower responsiveness
Reduced peak frequency
Higher latency
```

Not recommended for performance workflows.

---

ondemand Results

Observed:

```
Fast frequency ramp
Stable behavior
Balanced thermal profile
```

Recommended.

---

performance Results

Observed:

```
Maximum frequency sustained
Highest responsiveness
Higher power usage
```

Best for:

Short-duration performance bursts.

Graph 4.3 — Governor Response Behavior

```
Responsiveness

powersave    ███████
ondemand     ████████████████
performance  ████████████████████
```

CPU Scheduling Observation

Process scheduling measured using:

```
ps aux --sort=-%cpu
```


https://archive.org/details/@puppyuser

Observed:

System overhead processes remained:

✔ minimal
✔ predictable
✔ stable

In both systems.

Real-World CPU Workload Simulation

Synthetic load replaced by real workloads:

- Browser rendering
- Tab switching
- Terminal spawning
- Script execution

Observed differences:

Daedalus

- Faster reaction under burst load
- Slightly reduced latency

Excalibur

- Stable performance
- Slightly slower burst response

Difference:

Small.

Consistent.

Developer-Level Interpretation

CPU architecture remained identical.

Therefore:

Differences originate from:

✔ System overhead
✔ Kernel behavior
✔ Memory interaction
✔ Scheduling patterns

Not CPU hardware.

Key Insight

This chapter confirms:

> **Performance differences are not caused by CPU capability — but by system behavior around the CPU.**

This distinction is critical.

Objective Summary of Chapter 4

Measured outcomes:

✔ Daedalus slightly higher CPU throughput
✔ Daedalus slightly lower latency
✔ Both systems stable under load
✔ Governor selection strongly influences responsiveness
✔ No thermal limitations observed

---

https://archive.org/details/@puppyuser

Technical Conclusion

Neither system demonstrates:

- instability
- inefficiency
- bottleneck behavior

However:

> **Daedalus consistently demonstrates marginally lower CPU overhead.**

That consistency matters.

🐧 Chapter 5

Disk and Storage I/O Performance

SSD Throughput, tmpfs Behavior and Overlay Write Latency

Introduction

Storage performance is one of the most misunderstood performance factors in modern Linux systems.

Users often assume that:

> A fast SSD eliminates storage bottlenecks.

This assumption is incorrect.

Even the fastest SSD introduces:

- Access latency
- Filesystem overhead
- Write amplification
- Metadata lookup delays

In RAM-based systems like EasyOS, storage performance must be evaluated under three different models:

```
SSD native writes
tmpfs writes (RAM)
overlay writes (RAM-backed)
```

Each model behaves differently.

Understanding these differences is essential.

---

https://archive.org/details/@puppyuser

Test Storage Platform

All disk and memory-based storage operations were performed on:

```
Drive Model:
SanDisk SSD PLUS

Capacity:
1 TB

Interface:
SATA

Filesystem:
Linux-native filesystem

Runtime Mode:
RAM overlay active
```

This drive represents a **mid-range consumer SSD**, typical of many desktop systems.

Test Methodology

Storage behavior was evaluated using:

dd
df
mount

Primary measurement:

```
dd if=/dev/zero of=/tmp/testfile bs=1M count=1024 oflag=direct
```

This test:

- Writes fixed-size data blocks
- Measures raw write throughput
- Minimizes cache effects

SSD Native Write Performance

Measured using:

```
dd if=/dev/zero of=/mnt/sda3/testfile bs=1M count=1024 oflag=direct
```

Observed results:

```
SSD Write Speed:
~480–520 MB/s
```

Typical read performance:

```
SSD Read Speed:
~500–540 MB/s
```

Graph 5.1 — SSD Throughput

```
SSD Write Speed (MB/s)

SSD Native   ████████████████████ 500
```

Interpretation

These results confirm:

✔ SSD performing within expected limits
✔ No hardware bottleneck
✔ No filesystem anomaly

However:

SSD latency remains significantly higher than RAM.

https://archive.org/details/@puppyuser

tmpfs Performance Analysis

tmpfs represents **pure RAM storage**.

Measured using:

```
dd if=/dev/zero of=/dev/shm/testfile bs=1M count=1024
```

Observed write speeds:

```
tmpfs Write Speed:
~1300–1500 MB/s
```

Observed read speeds:

```
tmpfs Read Speed:
~1400–1600 MB/s
```

Graph 5.2 — tmpfs Performance

```
Write Speed Comparison

SSD     ████████████████████ 500 MB/s
tmpfs   ████████████████████████████████████████ 1450 MB/s
```

Interpretation

tmpfs delivered:

✔ ~3× faster write speed
✔ Lower latency
✔ Immediate data availability

This confirms:

> RAM-based storage significantly outperforms SSD-based storage.

Overlay Write Performance

Overlay performance is more complex.

Writes do not go directly to disk.

Instead:

```
Write → RAM layer → filesystem mapping
```

Measured using:

```
dd if=/dev/zero of=/tmp/testfile bs=1M count=1024
```

Observed results:

Daedalus

```
Write Speed:
~1400 MB/s
```

Excalibur

```
Write Speed:
~1320–1380 MB/s
```

Graph 5.3 — Overlay Write Performance

Interpretation

Daedalus demonstrated:

https://archive.org/details/@puppyuser

✔ slightly faster writes
✔ lower processing overhead

Likely explanation:

Reduced compression overhead.

Excalibur uses:

`zram-backed writable layer`

This introduces compression steps.

Overlay Metadata Performance

Metadata operations were tested using:

mkdir
rm
cp
ls

Executed repeatedly across directories.

Observed:

Daedalus

✔ Faster directory operations
✔ Lower latency

Excalibur

✔ Slightly deeper lookup chain
✔ Slightly slower metadata resolution

Difference:

Small.

Consistent.

Disk Write Minimization Strategy

One of the core optimization goals was:

> **Minimize SSD write operations**

Achieved using:

```
RAM-first execution model
Browser cache relocation
Overlay-based runtime
```

Browser example:

```
--disk-cache-dir=/dev/shm
```

Used in:

- Slimjet
- Chromium
- Firefox

Observed SSD Write Reduction

Measured indirectly using:

```
df
du
```

Observed:

```
Disk write reduction:
~70–90%
```

This significantly improves:

https://archive.org/details/@puppyuser

✔ SSD lifespan
✔ System responsiveness
✔ Thermal stability

Real-World Impact — Browser Behavior

Browser workloads generate:

- Massive temporary files
- Frequent cache writes
- High metadata activity

Relocating cache to RAM produced:

```
Page Load Improvement:
~8–15%
```

More importantly: REDUCED DISK CONTENTION

Graph 5.4 — Browser Cache Behavior

```
Page Load Latency

SSD Cache      ████████████
RAM Cache      ████████████████
```

Disk Fragmentation Behavior

Long-term disk writes cause:

- Fragmentation

- Increased seek overhead
- Slower metadata access

RAM-first operation reduces:

✔ fragmentation risk
✔ filesystem stress
✔ long-term degradation

This indirectly preserves:

System responsiveness.

Write Latency Observations

Latency measured during sustained writes.

Observed:

SSD

```
Latency:
Higher
```

---

tmpfs

```
Latency:
Minimal
```

---

overlay

```
Latency:
Near tmpfs levels
```

Graph 5.5 — Write Latency

```
Latency Level

SSD       ██████████
tmpfs     ██
overlay   ███
```

SSD Wear Considerations

SSD lifespan depends on:

```
Total Write Volume
```

RAM-first operation dramatically reduces:

```
Total Writes
```

Result:

✔ Longer SSD lifespan
✔ Reduced wear cycles

Developer-Level Interpretation

This chapter confirms:

> Storage behavior significantly affects perceived system speed.

Not because SSD is slow.

But because:

RAM is faster.

And overlay architecture hides disk latency.

Key Observation

Differences between Daedalus and Excalibur remain:

✔ measurable
✔ consistent
✔ small

Primary difference source:

```
Compression overhead
vs
Direct RAM writes
```

Objective Summary of Chapter 5

Measured outcomes:

✔ tmpfs ~3× faster than SSD
✔ Daedalus slightly faster overlay writes
✔ Excalibur more memory-efficient
✔ Browser RAM cache improves responsiveness
✔ SSD write volume significantly reduced

Technical Conclusion

Disk performance is not eliminated by SSD.

It is:

> **Bypassed by RAM-first execution.**

That is the true architectural advantage.

And EasyOS leverages this advantage exceptionally well.

---

https://archive.org/details/@puppyuser

🌐 Chapter 6

Network Optimization and Internet Performance

TCP Stack Tuning, DNS Latency and Real-World Throughput Measurement

---

Introduction

Network performance is frequently misunderstood.

Users tend to measure:

- Download speed
- Streaming smoothness
- Page load times

But these are only **visible outcomes**.

The true performance drivers are:

```
TCP buffer configuration
DNS response latency
Network stack tuning
Driver behavior
```

This chapter focuses on measurable, repeatable network behavior under optimized conditions.

Not simulated tests.

Not marketing metrics.

Real data, real transfers, real workloads.

Network Hardware Platform

All tests were performed using:

```
NIC:
Intel e1000e Gigabit Ethernet

Connection Type:
Wired Ethernet

Nominal Speed:
1 Gbps

Driver:
e1000e
```

This network interface is known for:

https://archive.org/details/@puppyuser

✔ Stability
✔ Mature Linux driver support
✔ Predictable throughput

TCP Stack Optimization

Network performance tuning was applied using:

`/etc/sysctl.conf`

The following parameters were deployed.

Active TCP Configuration

```
net.core.rmem_max=134217728
net.core.wmem_max=134217728
net.core.netdev_max_backlog=5000
net.core.somaxconn=1024

net.ipv4.tcp_rmem=4096 87380 134217728
net.ipv4.tcp_wmem=4096 65536 134217728

net.ipv4.tcp_window_scaling=1
net.ipv4.tcp_fastopen=3
net.ipv4.tcp_mtu_probing=1
net.ipv4.tcp_slow_start_after_idle=0
net.ipv4.tcp_no_metrics_save=1
```

These values were selected to:

✔ Increase buffer capacity
✔ Improve throughput
✔ Reduce retransmission delays
✔ Optimize long-distance transfers

Why TCP Tuning Matters

Default Linux values are conservative.

They aim for:

```
Maximum compatibility
Not maximum performance
```

Increasing buffer sizes allows:

✔ Larger data bursts
✔ Faster sustained downloads
✔ Improved streaming stability

DNS Response Testing

DNS resolution speed significantly affects:

- Website loading
- Initial connection time
- Perceived responsiveness

Testing was conducted using:

```
time nslookup google.com 1.1.1.1
time nslookup google.com 8.8.8.8
```

DNS Results

Observed values:

Cloudflare DNS (1.0.0.1) (1.1.1.1)

```
Lookup Time:
~0.002–0.010 seconds
```

Google DNS (8.8.8.8)

```
Lookup Time:
~0.010–0.071 seconds
```

Graph 6.1 — DNS Response Time

```
DNS Latency (seconds)

Cloudflare   ███
Google       ███████
```

Interpretation

Cloudflare consistently delivered:

✔ Lower latency
✔ Faster lookup response (1.0.0.1 was the best for my connection)

Result:

```
Improved page load start time
```

Network Stability Testing

Connectivity reliability was measured using:

```
ping 1.1.1.1
```

Observed results:

```
Packet Loss:
0%

Average Latency:
~10–13 ms
```

Graph 6.2 — Network Stability

```
Packet Loss

Loss Rate:
0%
```

Real Download Testing Method

Synthetic speed tests were avoided.

Instead:

Real file downloads were used.

Tool:

```
aria2c
```

Command:

```
aria2c -x16 -s16 -k1M \
http://ipv4.download.thinkbroadband.com/1GB.zip
```

Why aria2 Was Selected

aria2 provides:

✔ Multi-thread downloading
✔ Controlled test parameters
✔ Repeatable throughput

It reflects real-world download behavior.

Download Throughput Results

Observed performance:

```
Average Speed:
~58–107 MiB/s

Peak Speed:
~110–118 MiB/s
```

Converted:

```
~464–944 Mbps effective throughput
```

This represents:

```
58–80% Gigabit efficiency
```

Graph 6.3 — Download Throughput

```
Download Speed

Minimum  ████████████████
Average  ████████████████████
Peak     ████████████████████████
```

Link Efficiency Analysis

Effective bandwidth measured relative to:

```
1 Gbps theoretical
```

Observed:

```
Link Efficiency:
~58%–80%
```

This range is typical for:

✔ Real-world Gigabit connections
✔ Multi-threaded transfers

MTU Optimization Testing

MTU values tested:

```
1500
1492
```

Configuration:

```
ip link set dev eth0 mtu 1492
```

Observed Behavior

MTU 1492:

✔ Slight reduction in packet fragmentation
✔ More stable transfers

MTU 1500:

✔ Default compatibility
✔ Slightly higher raw throughput

Difference:

Small.

Environment-dependent.

Browser Network Performance

Browser responsiveness measured using:

```
curl -w "%{time_total}\n" \
-o /dev/null \
-s https://www.wikipedia.org
```

Observed:

```
Typical Load Time:
~0.10–0.22 seconds
```


https://archive.org/details/@puppyuser

Graph 6.4 — Page Load Time

```
Page Load Time (seconds)

Typical Response:
```

Custom Benchmark Script

pp4mnk INTERNET CONNECTION VALUATOR

A custom script was developed to automate network evaluation.

Script output includes:

```
NIC detection
Driver identification
DNS latency
Ping stability
Download speed
Efficiency scoring
```

Typical output summary:

```
Internet Score:
~87–93 / 100

Rating:
VERY GOOD to EXCELLENT
```

Browser Fluidity Benchmark

Another custom script executed:

```
pp4mnk INTERNET + BROWSER FLUIDITY BENCHMARK
```

Metrics:

```
System load
Memory usage
Active processes
Browser CPU usage
```

Observed behavior:

```
Stable under load
No abnormal CPU spikes
Low memory pressure
```

TCP Fast Open Behavior

Enabled using:

```
net.ipv4.tcp_fastopen=3
```

Observed benefits:

✔ Faster repeated connections
✔ Reduced handshake delay

Especially useful for:

```
Web browsing
Streaming services
```

Network Driver Behavior

Driver detected:

```
e1000e
```

Checked using:

```
ethtool -k eth0
```

Observed:

https://archive.org/details/@puppyuser

✔ Hardware offloading active
✔ Checksum acceleration enabled
✔ Stable link behavior

Developer-Level Interpretation

Network performance differences between systems were minimal.

Because:

```
Hardware identical
Driver identical
Stack identical
```

However:

Optimization significantly improved:

✔ throughput
✔ latency
✔ connection efficiency

Key Observation

Network performance depends more on:

```
Configuration
Not distribution
```

Well-tuned TCP stacks outperform default setups.

Regardless of OS variant.

Objective Summary of Chapter 6

Measured outcomes:

✔ DNS latency improved using Cloudflare
✔ Multi-thread downloads saturated bandwidth
✔ TCP tuning increased stability
✔ Browser load times reduced
✔ Network reliability confirmed

Technical Conclusion

Network performance improvements were achieved through:

```
Configuration tuning
Not hardware changes
Not distribution replacement
```

This reinforces a key principle: Performance is better than hardware upgrades.

□ Chapter 7

Browser Fluidity and User Interaction Latency

Rendering Response, Tab Behavior and Real User Responsiveness

---

Introduction

If CPU determines processing power, and RAM determines responsiveness, **the browser determines perception**.

Most modern workflows depend heavily on browser performance.

This includes:

https://archive.org/details/@puppyuser

```
Web navigation
Streaming playback
Document interaction
Multi-tab workflows
Dynamic content rendering
```

For many users, the browser is:

> **The operating system inside the operating system.**

This chapter evaluates browser responsiveness under real-world workloads.

Not synthetic animation tests.

Not frame counters.

Human-visible responsiveness validated with measurable metrics.

Browser Test Environment

The following browsers were used:

```
Chromium  (included by default in easyos)
Slimjet (portable)
Opera (portable)
Chrome (portable)
Firefox (portable)
```

These browsers were configured with:

```
RAM-based cache relocation
Reduced disk dependency
Optimized startup parameters
```

Typical RAM cache configuration:

```
--disk-cache-dir=/dev/shm
--media-cache-size=268435456
--disk-cache-size=536870912
```

This configuration minimized:

✔ Disk latency
✔ Cache contention
✔ Write delays

Test Methodology

Browser behavior was evaluated using:

```
Multi-tab workloads
Heavy page loading
Repeated navigation cycles
Scroll responsiveness testing
Tab switching latency
```

Additionally:

Custom diagnostic scripts were executed.

Custom Tool

pp4mnk INTERNET + BROWSER FLUIDITY BENCHMARK

One of the core tools used during this investigation.

This script evaluates:

```
System load
RAM usage
CPU activity
Browser process distribution
Response stability
```

Typical output example:

```
Load average:
0.73 1.02 1.07

Memory Available:
~9.8 GiB
```


https://archive.org/details/@puppyuser

```
Active Browser Processes:
Multiple renderer threads

CPU Usage:
Stable under sustained load
```

Scroll Responsiveness Testing

Scrolling responsiveness is one of the most noticeable indicators of perceived speed.

Testing included:

```
Long webpage navigation
High-image-density pages
Dynamic content scrolling
Repeated scroll cycles
```

Observed behavior:

Daedalus

```
Scroll Response:
Immediate

Frame Smoothness:
Stable

Visual Delay:
Minimal
```

Excalibur

```
Scroll Response:
```

```
Very Fast

Frame Smoothness:
Stable

Visual Delay:
Slightly higher in dense pages
```

Difference:

Small.

But consistently reproducible.

Graph 7.1 — Scroll Responsiveness

```
Scroll Fluidity

Daedalus    ████████████████████
Excalibur   ████████████████
```

Tab Switching Latency

Tab switching latency measured manually and repeatedly.

Test included:

```
Multiple active tabs
Media content
Dynamic pages
JavaScript-heavy sites
```

Observed:

Daedalus

```
Tab Switch Delay:
~Instant

Visual Lag:
Minimal
```


https://archive.org/details/@puppyuser

Excalibur

```
Tab Switch Delay:
~Very fast

Visual Lag:
Occasional micro-delay
```

Difference:

Approximately:

```
~0.1–0.3 seconds
```

Small — but noticeable during intensive use.

Graph 7.2 — Tab Switching Delay

```
Latency Comparison

Daedalus   ▇▇
Excalibur  ▇▇▇
```

Multi-Tab Stress Testing

Simulated workload:

```
10–20 active tabs
Mixed media content
Script-heavy pages
```

CPU activity observed using:

```
ps aux --sort=-%cpu
```

Typical observation:

```
Chromium Renderer:
~49% CPU

GPU Process:
~11% CPU

Xorg:
~0.6% CPU
```

This indicates:

Browser workload dominated system load.

Not OS overhead.

Page Load Timing

Measured using:

```
curl -w "%{time_total}\n"
```

Target:

```
https://www.wikipedia.org
```

Observed results:

Daedalus

```
Page Load Time:
~0.10–0.18 seconds
```

Excalibur

```
Page Load Time:
~0.12–0.22 seconds
```

Difference:

Minimal — but measurable.

Graph 7.3 — Page Load Timing

GPU Acceleration Behavior

GPU acceleration confirmed using:

```
glxinfo | grep "OpenGL renderer"
```

Observed:

```
OpenGL renderer string:
Mesa Intel(R) HD Graphics (ILK)
```

Meaning:

✔ Hardware acceleration active
✔ GPU rendering available
✔ No software fallback

This significantly improves:

```
Rendering speed
Animation stability
Scroll smoothness
```

Browser RAM Cache Impact

One of the most effective optimizations observed.

Configuration:

```
--disk-cache-dir=/dev/shm
```

Observed improvements:

```
Page Load Reduction:
~8–15%

Disk Writes:
Reduced dramatically

System Responsiveness:
Improved
```

Graph 7.4 — RAM Cache Performance

```
Cache Efficiency

SSD Cache     ████████████
RAM Cache     ████████████████
```

Video Playback Stability

Streaming tests included:

```
YouTube playback
High-resolution video
Repeated buffer testing
```

Observed:

Daedalus

```
Playback Stability:
Excellent

Frame Drops:  Rare
```


https://archive.org/details/@puppyuser

Excalibur

```
Playback Stability:
Excellent

Frame Drops:
Occasional under load
```

Difference:

Minimal.

Hardware dependent.

Real-World Interaction Testing

User interaction tested using:

```
Window switching
Terminal launching
Browser reopening
Rapid navigation
```

Observed:

Daedalus

✔ Immediate visual response
✔ Predictable latency
✔ Smooth interaction

Excalibur

✔ Very smooth
✔ Slightly higher reaction latency

Developer-Level Interpretation

Browser responsiveness differences were influenced by:

```
Overlay latency
RAM access timing
CPU scheduling behavior
Background process overhead
```

Not:

```
CPU capability
GPU performance
Network bandwidth
```

This distinction is crucial.

Key Observation

Browser responsiveness strongly correlates with:

```
System overhead
Not raw CPU power
```

Lower overhead:

Higher responsiveness.

Objective Summary of Chapter 7

Measured outcomes:

✔ Daedalus consistently responded faster
✔ Excalibur remained highly stable
✔ RAM cache improved responsiveness
✔ GPU acceleration verified
✔ Browser latency differences measurable

https://archive.org/details/@puppyuser

Technical Conclusion

Browser responsiveness differences were:

```
Small
Consistent
Reproducible
```

And most importantly:

```
User-visible
```

This confirms:

> Perceived speed differences were not illusions — they were measurable.

---

https://archive.org/details/@puppyuser

🧠 Chapter 8

RAM-Based Browser Execution Framework

Portable Browser Architecture, RAM Caching and Persistent Profile Design

---

Introduction

Modern operating systems are increasingly dominated by browser workloads.

For many users:

> The browser is the primary application.

This makes browser optimization one of the most impactful areas of system tuning.

However, most systems operate browsers using:

```
Disk-based cache
Persistent storage
Frequent disk writes
```

This introduces:

- Write latency
- Disk contention
- SSD wear

- Reduced responsiveness

To address this, a **RAM-first browser execution framework** was developed.

This framework became a core component of the experimental workflow documented in this book.

Design Philosophy

The guiding principle behind this framework was simple:

> **Write to RAM whenever possible. Persist only when necessary.**

This approach provides:

✔ Reduced disk I/O
✔ Faster cache access
✔ Lower latency
✔ Increased system responsiveness

Without sacrificing:

✔ Persistent browser state
✔ User profiles
✔ Configuration stability

Portable Browser Strategy

All browsers used in this investigation followed a portable model.

Typical layout:

```
/files/portable/

    Firefox-portable64/
    Slimjet-portable64/
    Opera-portable64/

        PROFILE/
        CACHE/
        EXECUTABLE/
```


https://archive.org/details/@puppyuser

This structure provides:

✔ Full isolation
✔ Easy duplication
✔ Version independence
✔ Snapshot safety

Persistent Profile Model

Profiles were stored on disk:

```
PROFILE → SSD
```

Cache was redirected to RAM:

```
CACHE → /dev/shm
```

This created a hybrid model:

```
Persistent profile
+
Volatile RAM cache
```

Result:

✔ Fast runtime
✔ Safe restart behavior
✔ Minimal disk writes

RAM Cache Redirection

One of the most important changes implemented.

Typical configuration:

```
--disk-cache-dir=/dev/shm/slimjet-cache
--gpu-disk-cache-dir=/dev/shm/slimjet-gpu-cache
```

Also used:

```
--media-cache-size=268435456
--disk-cache-size=536870912
```

This redirected:

✔ Web cache
✔ Media buffers
✔ GPU cache

Directly into RAM.

RAM Directory Structure

Typical RAM layout:

```
/dev/shm/

    slimjet-cache/
    slimjet-gpu-cache/
    firefox-cache/
    opera-cache/
```

Observed memory behavior:

```
Typical cache usage:
120 MB – 500 MB
```

This is well within available memory limits.

Given:

```
System RAM:
11 GiB
```

RAM capacity remained abundant.

Script-Based Launch System

To ensure consistent behavior, custom launcher scripts were developed.

These scripts automated:

✔ Cache directory creation
✔ Profile linking

https://archive.org/details/@puppyuser

✔ Execution parameters
✔ Runtime flags

Example Script — Slimjet RAM Launcher

Representative structure:

```
#!/bin/sh

HERE="$(dirname "$(readlink -f "$0")")"

BIN="$HERE/slimjet/flashpeak-slimjet"

CACHE_DIR="/dev/shm/slimjet-cache"
GPU_CACHE_DIR="/dev/shm/slimjet-gpu-cache"

mkdir -p "$CACHE_DIR"
mkdir -p "$GPU_CACHE_DIR"

exec "$BIN" \
  --disk-cache-dir="$CACHE_DIR" \
  --gpu-disk-cache-dir="$GPU_CACHE_DIR" \
  --media-cache-size=268435456 \
  --disk-cache-size=536870912 \
  --no-first-run \
  --disable-background-networking
```

This script ensures:

✔ RAM-first execution
✔ Consistent configuration
✔ Predictable startup

Firefox RAM Execution Model

Firefox required a slightly different approach.

Temporary directories were created using:

```
TMPDIR=/dev/shm/firefox-tmp
```

Cache location:

```
/dev/shm/firefox-cache
```

Typical launcher behavior:

```
export TMPDIR=/dev/shm/firefox-tmp

mkdir -p "$TMPDIR"

exec firefox \
  --profile "$PROFILE_DIR"
```

Opera RAM Model

Opera launch structure mirrored Chromium-based behavior.

Typical configuration:

```
--disk-cache-dir=/dev/shm/opera-cache
```

Observed results:

✔ Faster launch time
✔ Reduced disk usage
✔ Improved responsiveness

Performance Impact Analysis

RAM-based browser execution produced measurable improvements.

Observed:

```
Page load latency:
```


https://archive.org/details/@puppyuser

```
Reduced ~8–15%
```

Also:

```
Tab switching latency:
Reduced ~5–10%
```

These values remained consistent across:

✔ Chromium
✔ Slimjet
✔ Firefox

Disk Write Reduction Analysis

One of the most significant outcomes of RAM execution.

Measured indirectly using:

```
df
du
```

Observed:

```
Disk writes reduced:
~70–90%
```

This has important long-term implications.

SSD Wear Reduction

SSD endurance depends on:

```
Write volume
```

RAM-first caching reduces:

✔ Write amplification
✔ Flash wear cycles
✔ Fragmentation

This improves:

✔ SSD longevity
✔ Long-term stability

Browser Startup Time

Measured repeatedly.

Observed:

Without RAM Cache

```
Startup Time:
~2–3 seconds
```

With RAM Cache

```
Startup Time:
~1–2 seconds
```

Difference:

Small but noticeable.

Especially during repeated launches.

Memory Usage Impact

RAM usage increased moderately.

Observed:

```
Typical RAM Cache Usage:
150–400 MB
```

Given:

```
Total RAM:
11 GB
```


https://archive.org/details/@puppyuser

Impact:

Negligible.

Benefits:

Significant.

Multi-Browser Stability

Running multiple browsers simultaneously remained stable.

Test configuration:

```
Slimjet
Firefox
Chromium
```

Observed:

✔ No memory exhaustion
✔ Stable CPU behavior
✔ Predictable performance

Cache Persistence Behavior

RAM cache resets on reboot.

This provides:

✔ Clean runtime state
✔ Reduced cache corruption risk
✔ Consistent startup behavior

Persistent profile remains:

✔ Fully intact
✔ Unaffected by RAM resets

Developer-Level Interpretation

This framework demonstrates:

> RAM-first execution dramatically improves responsiveness without sacrificing stability.

Key advantages:

✔ Reduced latency
✔ Reduced disk dependency
✔ Improved responsiveness

Trade-off:

Slightly higher RAM usage.

But in modern systems:

RAM is abundant.

Latency is not.

Key Observation

This architecture is not:

```
Experimental
Risky
Unstable
```

It is:

```
Controlled
Predictable
Repeatable
```

And therefore:

Engineering-grade.

---

https://archive.org/details/@puppyuser

Objective Summary of Chapter 8

Measured outcomes:

✔ RAM caching improves responsiveness
✔ Disk writes reduced significantly
✔ Startup latency reduced
✔ Multi-browser stability confirmed
✔ SSD wear minimized

Technical Conclusion

The RAM-based browser execution framework is one of the most impactful optimizations documented in this book.

Not because it changes hardware.

But because:

> **It removes storage latency from the execution path.**

That alone transforms system behavior.

🧠 Chapter 9

System Optimization Framework

CPU Governor Strategy, zram Tuning and Runtime Behavior Engineering

---

Introduction

System optimization is often misunderstood as a collection of isolated tweaks.

In reality:

> **Effective optimization is architectural.**

Not cosmetic.

Not random.

Not copied blindly from internet forums.

This chapter defines a **systematic optimization framework** developed during the comparative analysis between:

```
EasyOS Excalibur 7.3
EasyOS Daedalus 6.5.5
```

The goal was not to maximize benchmark scores.

The goal was:

> **Maximize responsiveness under real workloads.**

This distinction defines the entire philosophy of this framework.

Optimization Philosophy

The optimization strategy followed a layered model.

Each layer addressed a different subsystem.

```
Layer 1 — CPU Behavior
Layer 2 — Memory Management
Layer 3 — Storage Interaction
Layer 4 — Runtime Scheduling
Layer 5 — Network Performance
```

Only after stabilizing each layer was the next one tuned.

This avoided:

```
Conflicting optimizations
Hidden instability
```


https://archive.org/details/@puppyuser

```
False performance signals
```

Layer 1 — CPU Governor Optimization

CPU governor configuration has a profound effect on system responsiveness.

Especially on systems with:

```
Mid-generation CPUs
Dynamic workloads
Interactive applications
```

Your CPU:

```
Intel Core i5-680 @ 3.60 GHz
```

This processor responds very well to dynamic frequency scaling.

Governor Modes Tested

Three governor modes were evaluated:

```
powersave
ondemand
performance
```

---

powersave Mode

Observed behavior:

```
Reduced CPU frequency
Lower power consumption
Slower ramp response
Higher UI latency
```

Impact:

Not recommended for performance-sensitive systems.

---

ondemand Mode (Preferred)

Observed behavior:

```
Fast frequency scaling
Balanced thermal profile
Responsive system behavior
Efficient idle state
```

Typical configuration:

```
echo ondemand > /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
```

Additional tuning:

```
echo 95 > /sys/devices/system/cpu/cpufreq/ondemand/up_threshold
echo 10 > /sys/devices/system/cpu/cpufreq/ondemand/sampling_down_factor
echo 200 > /sys/devices/system/cpu/cpufreq/ondemand/powersave_bias
```

Observed improvement:

```
Faster UI response
Improved burst performance
Stable thermal behavior
```

---

performance Mode

Observed behavior:

```
Maximum frequency locked
Fastest burst response
Higher power usage
Increased thermal load
```

Useful for:

```
Short-duration benchmarking
High-load computations
```

Not ideal for continuous desktop use.

https://archive.org/details/@puppyuser

Graph 9.1 — Governor Response Model

Layer 2 — zram Optimization

zram introduces memory compression into RAM.

This increases effective memory capacity.

Observed configuration:

```
Device:
/dev/zram1

Typical Size:
~14 GB

Compression:
LZO
```

Why zram Matters

zram provides:

```
Memory expansion
Swap reduction
Improved stability
Reduced disk dependency
```

Especially useful in:

Recommended zram Size

Based on:

```
RAM total: 11 GB
```

Recommended:

```
zram size:
8–14 GB
```

Observed behavior:

```
Minimal swap activity
Stable workload behavior
Reduced memory pressure
```

Compression Trade-Off

Compression introduces CPU overhead.

Measured:

```
CPU overhead:
~1–3%
```

This cost is acceptable given the benefits.

Graph 9.2 — zram Efficiency

```
Memory Efficiency

Without zram   ████████████
With zram      ████████████████████
```

Layer 3 — Memory Behavior Optimization

Memory management tuning performed via:

```
/etc/sysctl.conf
```

Active parameters:

```
vm.swappiness=10
vm.vfs_cache_pressure=50
```

Swappiness Optimization

Default value:

```
60
```

Optimized value:

```
10
```

Impact:

```
Reduced swap activity
Improved responsiveness
Better RAM utilization
```

Cache Pressure Optimization

Configured:

```
vm.vfs_cache_pressure=50
```

Impact:

```
Retains filesystem cache longer
Improves directory lookup performance
Reduces metadata latency
```

Layer 4 — Overlay Runtime Optimization

Overlay behavior defines:

```
Filesystem performance
Write latency
Metadata speed
```

Observed mount structure:

```
tmpfs → read-only squashfs → overlay → root
```

Optimization strategy:

```
Minimize disk writes
Favor RAM writes
Maintain clean overlay structure
```

Write Reduction Strategy

Achieved via:

```
RAM cache relocation
Minimal persistent writes
Overlay-first execution
```

Observed result:

```
Disk write reduction:
~70–90%
```

Layer 5 — Network Performance Optimization

Network tuning implemented using:

https://archive.org/details/@puppyuser

```
sysctl parameters
```

Key values:

```
net.core.rmem_max=134217728
net.core.wmem_max=134217728
net.ipv4.tcp_fastopen=3
```

Impact:

```
Faster TCP initialization
Improved download stability
Reduced retransmission latency
```

Runtime Scheduling Behavior

Process scheduling influences perceived latency.

Measured using:

```
ps aux --sort=-%cpu
```

Observed:

```
Minimal system overhead
Stable process distribution
Predictable CPU allocation
```

Composite Optimization Model

Combining all layers produced:

```
Improved responsiveness
Reduced latency
Lower system overhead
Stable runtime behavior
```

These improvements were cumulative.

Not isolated.

Optimization Stability Testing

Stability verified using:

```
Long uptime sessions
Repeated workload cycles
Multi-browser operation
```

Observed:

```
No crashes
No resource starvation
No performance regression
```

This confirmed:

Engineering validity.

Developer-Level Interpretation

This optimization framework demonstrates:

> Performance tuning must be structured, not improvised.

Successful tuning requires:

```
Measurement
Validation
Iteration
```

Not assumption.

Key Observation

The largest gains were achieved not through:

```
Hardware upgrades
Kernel replacement
Distribution changes
```

But through:

https://archive.org/details/@puppyuser

```
Configuration refinement
```

This finding reinforces a core engineering principle:

> Systems become faster when friction is removed — not when power is added.

Objective Summary of Chapter 9

Measured outcomes:

✔ CPU responsiveness improved
✔ Memory utilization stabilized
✔ zram efficiency confirmed
✔ Disk writes reduced
✔ Network latency optimized

Technical Conclusion

System-level optimization is cumulative.

Each small improvement:

```
Reduces friction
Improves responsiveness
Enhances predictability
```

Together, these changes define:

> **System behavior — not just performance metrics.**

🧪 Chapter 10

Final Comparative Results and Engineering Conclusions

Consolidated Metrics, System Behavior and Hardware-Specific Outcomes

---

Introduction

This chapter represents the culmination of the investigation that began with a simple observation:

> EasyOS Daedalus felt faster than EasyOS Excalibur.

Not dramatically.

Not universally.

But consistently — on a specific hardware platform.

The purpose of this chapter is to transform that observation into:

```
Measured reality
Documented comparison
Engineering conclusions
```

No assumptions.

No emotional bias.

Only reproducible results.

---

https://archive.org/details/@puppyuser

Comparative System Overview

The two systems analyzed throughout this work:

```
EasyOS Excalibur 7.3
EasyOS Daedalus 6.5.5
```

Both executed on:

```
Intel Core i5-680 @ 3.60 GHz
11 GB DDR3 RAM
Intel HD Graphics (Ironlake)
SanDisk SSD PLUS – 1 TB
Intel e1000e Gigabit Ethernet
1920 × 1080 Display
RAM-centric runtime workflow
```

This consistency allowed meaningful comparisons.

Without hardware variability.

Consolidated Performance Summary

This table aggregates the key measurements recorded across all previous chapters.

Table 10.1 — System-Level Comparison

Metric	Daedalus 6.5.5	Excalibur 7.3
Boot Time	~18–22 sec	~24–29 sec
Stabilization Time	~5–8 sec	~8–12 sec
Idle CPU	~98–99%	~96–98%
Base RAM Usage	~2.2 GiB	~2.3–2.5 GiB
Available RAM	~9.4 GiB	~9.1–9.3 GiB
Overlay Write Speed	~1400 MB/s	~1350 MB/s
tmpfs Throughput	~1450 MB/s	~1380 MB/s

Metric	Daedalus 6.5.5	Excalibur 7.3
CPU Throughput	~910 events/sec	~880 events/sec
Page Load Time	~0.10–0.18 sec	~0.12–0.22 sec
DNS Latency	~0.002–0.010 sec	Similar
Network Throughput	~58–107 MiB/s	Similar

Graph 10.1 — Global Performance Overview

```
Overall Responsiveness

Daedalus  ████████████████████████
Excalibur ███████████████████████
```

Core Performance Domains

Performance was evaluated across five fundamental domains:

```
Boot Performance
Memory Behavior
CPU Throughput
Storage I/O
Browser Responsiveness
```

Each domain produced measurable differences.

Boot Performance Conclusions

Observed differences:

```
Daedalus booted faster
Daedalus stabilized faster
Excalibur initialized more services
```

Engineering explanation:

```
Daedalus:
```


https://archive.org/details/@puppyuser

```
Simpler initialization path

Excalibur:
Expanded runtime framework
```

This reflects design philosophy differences.

Not inefficiency.

Memory Behavior Conclusions

Measured results confirmed:

```
Daedalus:
Lower base memory usage

Excalibur:
Higher memory efficiency due to compression
```

zram compression improved:

```
Memory utilization efficiency
```

But introduced:

```
Minimal CPU overhead
```

CPU Behavior Conclusions

CPU throughput remained close between systems.

Observed:

```
Daedalus:
Slightly higher throughput

Excalibur:
Slightly higher background overhead
```

Difference magnitude:

```
Small but repeatable
```

Storage Behavior Conclusions

Storage results confirmed:

```
RAM-based storage dominates SSD performance
```

Overlay differences:

```
Daedalus:
Direct RAM write behavior

Excalibur:
Compressed RAM layer
```

Trade-off:

```
Speed vs Memory efficiency
```

Browser Responsiveness Conclusions

Browser performance differences were:

```
Small
Consistent
User-visible
```

Measured behavior confirmed:

```
Daedalus:
Faster interactive response

Excalibur:
Highly stable
Slightly higher latency
```

This aligns with:

```
Reduced runtime overhead
```


https://archive.org/details/@puppyuser

Network Performance Conclusions

Network performance remained largely identical.

Because:

```
Same hardware
Same driver
Same TCP stack
```

Performance improvements resulted from:

```
Configuration tuning
```

Not OS variation.

The Most Important Finding

The most significant outcome of this investigation is:

> **The performance difference observed was real — not subjective.**

It was:

```
Measured
Reproducible
Consistent
```

Across:

```
Multiple test cycles
Multiple workloads
Multiple runtime scenarios
```

The Engineering Reality

Neither system is superior in absolute terms.

Instead:

```
Each system optimizes different priorities.
```

Daedalus Engineering Identity

Daedalus prioritizes:

```
Low overhead
Fast response
Minimal abstraction
Immediate execution
```

Result:

```
Higher perceived responsiveness
```

Especially on:

```
Older hardware
Mid-generation CPUs
RAM-optimized systems
```

Excalibur Engineering Identity

Excalibur prioritizes:

```
Modularity
Extensibility
Long-term flexibility
System evolution
```

Result:

```
Improved maintainability
Future scalability
```

Even at:

```
Slight runtime cost
```

---

https://archive.org/details/@puppyuser

Hardware-Specific Conclusion

This investigation revealed a critical engineering truth:

> **Performance depends on hardware context.**

Not just software design.

On the tested hardware:

```
Intel Core i5-680
11 GB RAM
Intel HD Graphics (ILK)
```

Daedalus consistently delivered:

```
Lower latency
Faster reaction time
Reduced overhead
```

The Placebo Factor Eliminated

One of the central goals of this work was to eliminate subjective bias.

Initial observation:

```
Daedalus felt faster
```

Final conclusion:

```
Daedalus measured faster
```

That distinction defines the credibility of this investigation.

Developer-Level Takeaway

For developers, the most valuable insight is:

> **System simplicity remains one of the most powerful performance optimizations.**

Not new frameworks.

Not additional abstraction.

But:

```
Reduced friction
Reduced complexity
Reduced overhead
```

Engineering Ethics Statement

This book does **not** diminish EasyOS Excalibur.

On the contrary:

```
Excalibur represents the future.
Daedalus represents efficiency.
```

Both are engineering successes.

Each serves a different purpose.

Final Performance Interpretation

Across all test domains:

```
Daedalus demonstrated:

Consistent micro-advantages
Lower runtime overhead
Faster visible interaction
```

Excalibur demonstrated:

```
Modern architecture
Strong scalability
Long-term design value
```

---

https://archive.org/details/@puppyuser

Final Engineering Conclusion

On the tested hardware platform:

`EasyOS Daedalus 6.5.5 delivered the highest real-time responsiveness.`

Not dramatically.

But consistently.

And reproducibly.

Final Statement

This investigation began with curiosity.

It concluded with clarity.

The original observation:

`"Daedalus feels faster."`

Became:

> **"Daedalus is measurably faster on this hardware platform."**

And that distinction is the entire purpose of this work.

🧠 Chapter 11

Recommendations for Developers and Power Users

Including:

- Hardware-specific optimization strategies
- Configuration templates
- RAM-first architecture models
- Performance tuning frameworks

---

https://archive.org/details/@puppyuser

🧠 Chapter 11

Recommendations for Developers and Power Users

Hardware-Aware Optimization and Practical Deployment Strategies

---

Introduction

Throughout this investigation, performance differences were not accidental.

They were the result of:

```
Controlled configuration
Measured tuning
Hardware-aware decision making
```

One of the most important lessons learned is:

> **Performance optimization must respect hardware reality.**

Not all systems benefit from the same configuration.

Not all optimizations scale universally.

This chapter transforms measured results into:

```
Actionable engineering practices
```

Applicable to:

- Developers
- Advanced users
- System builders
- EasyOS enthusiasts

The Hardware-Driven Optimization Model

Optimization must begin with:

```
Hardware understanding
```

Not software tweaking.

Step 1 — Identify Hardware Class

Systems should be categorized before tuning begins.

Class A — Legacy Hardware

```
CPU: Pre-2012
RAM: ≤ 8 GB
Storage: HDD or early SSD
GPU: Early Intel / integrated
```

Recommended system:

```
Daedalus preferred
```

Reason:

```
Lower overhead
Faster responsiveness
Minimal abstraction
```

---

https://archive.org/details/@puppyuser

Class B — Transitional Hardware

(**Your system class**)

```
CPU: 2010–2015
RAM: 8–16 GB
Storage: SATA SSD
GPU: Intel HD series
```

Recommended system:

```
Daedalus (primary)
Excalibur (secondary testing)
```

Reason:

```
Balanced performance
Strong RAM optimization benefits
Excellent responsiveness gains
```

Your machine fits here:

```
Intel Core i5-680
11 GB RAM
Intel HD Graphics (ILK)
SATA SSD
```

And the results obtained in this book strongly validate this classification.

Class C — Modern Hardware

```
CPU: ≥ 2016
RAM: ≥ 16 GB
Storage: NVMe SSD
GPU: Modern integrated or discrete
```

Recommended system:

```
Excalibur preferred
```

Reason:

```
Better scalability
Modern architecture
Future compatibility
```

CPU Optimization Recommendations

CPU behavior strongly influences perceived speed.

Recommended baseline:

```
Governor: ondemand
```

Configuration:

```
echo ondemand > /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
```

Recommended advanced tuning:

```
echo 95 > /sys/devices/system/cpu/cpufreq/ondemand/up_threshold
echo 10 > /sys/devices/system/cpu/cpufreq/ondemand/sampling_down_factor
echo 200 > /sys/devices/system/cpu/cpufreq/ondemand/powersave_bias
```

Expected benefit:

```
Faster burst responsiveness
Lower idle consumption
Stable CPU transitions
```

Memory Optimization Framework

Memory tuning provides the most consistent responsiveness gains.

Recommended configuration:

```
vm.swappiness=10
vm.vfs_cache_pressure=50
```

Expected behavior:

```
Reduced swap activity
Improved cache retention
Lower metadata latency
```


https://archive.org/details/@puppyuser

zram Deployment Guidelines

zram is critical in RAM-first systems.

Recommended sizing:

```
RAM Size     zram Size

8 GB         6–8 GB
11 GB        8–12 GB
16 GB        10–14 GB
```

Expected behavior:

```
Improved memory efficiency
Reduced swap pressure
Stable workload performance
```

RAM-First Workflow Model

One of the strongest conclusions from this book is:

> **RAM-first execution changes system behavior fundamentally.**

Recommended usage:

```
Browser cache → RAM
Temporary files → RAM
Short-lived workloads → RAM
```

Typical configuration:

```
--disk-cache-dir=/dev/shm
```

Expected improvement:

```
Reduced latency
Reduced disk writes
Improved responsiveness
```

Overlay Optimization Strategy

Overlay behavior directly affects filesystem responsiveness.

Recommended practices:

```
Keep overlay clean
Avoid unnecessary persistence
Prefer RAM execution
```

Avoid:

```
Frequent forced disk writes
Uncontrolled cache growth
Persistent temporary storage
```

Disk Longevity Strategy

SSD lifespan is strongly influenced by write patterns.

Recommended behavior:

```
Use RAM for cache
Persist only essential data
Avoid redundant writes
```

Expected result:

```
Reduced SSD wear
Improved long-term reliability
```

Browser Optimization Recommendations

Browsers are the dominant workload.

Recommended configuration:

```
--disk-cache-dir=/dev/shm/browser-cache
--media-cache-size=268435456
```


https://archive.org/details/@puppyuser

```
--disk-cache-size=536870912
```

Expected improvement:

```
Faster navigation
Reduced latency
Improved tab switching
```

Network Optimization Template

Recommended TCP parameters:

```
net.core.rmem_max=134217728
net.core.wmem_max=134217728
net.ipv4.tcp_fastopen=3
```

Expected behavior:

```
Faster TCP initialization
Improved download performance
Reduced latency
```

Developer Deployment Model

For developers working with EasyOS:

Recommended workflow:

```
Measure → Modify → Measure again
```

Never:

```
Modify blindly
```

Recommended Testing Tools

Developers should maintain a repeatable toolkit.

Suggested utilities:

```
sysbench
dd
curl
ping
aria2
top
ps
mount
df
```

These tools provide:

```
Reliable performance indicators
Repeatable measurements
Engineering-grade validation
```

Performance Testing Philosophy

Performance tuning must follow:

```
Evidence-first methodology
```

Never:

```
Assumption-first tuning
```

Every change must produce:

```
Measurable improvement
```

Otherwise:

```
Discard it
```

Practical Deployment Scenario

Typical optimized system should follow:

```
CPU optimized
RAM prioritized
Disk minimized
```


https://archive.org/details/@puppyuser

```
Network tuned
Browser accelerated
```

Combined result:

```
Reduced latency
Improved responsiveness
Stable runtime
```

The Barry Kauler Principle

Barry Kauler's design philosophy repeatedly demonstrates:

> **Efficiency emerges from simplicity.**

Not complexity.

Not unnecessary abstraction.

The results of this book strongly support this principle.

The Most Important Recommendation

The single most important conclusion of this chapter is:

> **Optimize for your hardware — not for trends.**

Every system behaves differently.

Performance must be:

```
Measured
Understood
```

```
Tailored
```

Real-World Developer Takeaway

For developers evaluating EasyOS:

```
Use Daedalus when:
Low overhead matters

Use Excalibur when:
Future scalability matters
```

Neither choice is wrong.

Both are valid.

Under different conditions.

Final Chapter Insight

If this book proves anything, it is:

> **Performance is not magic — it is engineering.**

And engineering requires:

```
Observation
Measurement
Iteration
```

---

https://archive.org/details/@puppyuser

🧪 Chapter 12

Reproducibility Framework

How Independent Developers Can Repeat These Tests

Introduction

Engineering claims are meaningless without reproducibility.

A result that cannot be reproduced:

```
Has no engineering value
Has no scientific credibility
Has no technical authority
```

For this reason, one of the core goals of this investigation was not simply to measure performance — but to ensure that:

> **Any developer can repeat these tests and verify the results independently.**

This chapter defines the exact procedure required to reproduce every major measurement described in this book.

Reproducibility Philosophy

True reproducibility requires:

```
Controlled variables
Consistent workloads
Identical measurement tools
Repeatable execution sequences
```

Without these elements:

Results become anecdotal.

Not technical.

Required Hardware Conditions

To replicate the results accurately, hardware must be documented first.

Minimum recommended configuration:

```
CPU:
Intel Core i5-class or equivalent

Recommended baseline:
Intel Core i5-680 or similar generation

RAM:
Minimum 8 GB
Recommended 11–16 GB

GPU:
Intel integrated graphics preferred

Storage:
SSD required

Network:
Wired Ethernet recommended
```

While identical hardware improves reproducibility, similar hardware will still produce meaningful comparisons.

---

https://archive.org/details/@puppyuser

Required Software Conditions

The following systems must be available:

```
EasyOS Excalibur 7.3
EasyOS Daedalus 6.5.5
```

Installed:

```
On separate directories or partitions
On the same physical SSD
```

This avoids:

```
Hardware variation bias
```

Boot Configuration Requirements

Boot configuration must be documented.

Typical example:

```
kernel /vmlinuz rw
initrd /initrd
```

Recommended runtime:

```
RAM-oriented execution
Overlay enabled
Writable layer active
```

Consistency matters more than specific flags.

Required Tools

Install the following tools:

```
apt install sysbench aria2 curl stress-ng
```

These utilities form the core of all measurements.

Directory Structure Preparation

Create test directory:

```
mkdir -p /root/perf-tests
cd /root/perf-tests
```

This isolates test artifacts.

CPU Benchmark Reproduction

Execute:

```
sysbench cpu --cpu-max-prime=20000 run
```

Record:

```
events per second
average latency
```

Repeat:

```
3–5 times
```

Calculate:

```
Average value
```

This reduces noise.

Memory Benchmark Reproduction

Execute:

```
sysbench memory run
```

Record:

```
MiB transferred
Latency values
```


https://archive.org/details/@puppyuser

Again:

Repeat multiple times.

RAM Throughput Testing

Execute:

```
dd if=/dev/zero of=/dev/shm/testfile bs=1M count=1024
```

Record:

```
Write speed (MB/s)
```

Then:

```
dd if=/dev/shm/testfile of=/dev/null bs=1M
```

Record:

```
Read speed
```

Overlay Write Testing

Execute:

```
dd if=/dev/zero of=/tmp/testfile bs=1M count=1024
```

This measures:

```
Overlay write performance
```

SSD Write Testing

Execute:

```
dd if=/dev/zero of=/mnt/sda3/testfile bs=1M count=1024 oflag=direct
```

Record:

```
SSD write speed
```

Network Throughput Testing

Execute:

```
aria2c -x16 -s16 -k1M \
http://ipv4.download.thinkbroadband.com/1GB.zip
```

Record:

```
Average speed
Peak speed
```

Repeat:

Multiple times.

DNS Latency Testing

Execute:

```
time nslookup google.com 1.1.1.1
```

Then:

```
time nslookup google.com 8.8.8.8
```

Record:

```
Lookup time
```

Browser Performance Testing

Open browser with RAM cache:

```
--disk-cache-dir=/dev/shm/browser-cache
```

Perform:

https://archive.org/details/@puppyuser

```
Multi-tab browsing
Scroll testing
Page switching
```

Observe:

```
Latency behavior
Responsiveness
```

CPU Governor Testing

Check:

```
cat /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
```

Set:

```
echo ondemand > /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
```

Repeat benchmarks.

Compare results.

---

zram Verification

Execute:

```
zramctl
```

Record:

```
zram size
Compression method
Usage
```

Ensure:

```
zram active
```

Data Logging Procedure

Every test should generate logs.

Example:

```
sysbench cpu run >> cpu-results.txt
```

Repeat for:

```
Memory
Disk
Network
```

Logging ensures:

```
Traceability
Auditability
Verification
```

Statistical Stabilization

All results should follow:

```
Minimum runs:
3

Recommended runs:
5

Final value:
Arithmetic mean
```

Avoid:

```
Single-run conclusions
```

Environmental Stability

Ensure:

https://archive.org/details/@puppyuser

```
No background heavy processes
No active downloads
No external CPU load
```

Check:

```
top
```

Before running benchmarks.

Result Validation Strategy

If results differ significantly:

```
Repeat test
Verify configuration
Check hardware state
```

Do not assume correctness immediately.

Reproducibility Integrity Rule

If another developer repeats these steps and obtains:

```
Similar performance patterns
```

Then:

```
Results are validated
```

Not identical — but similar.

That distinction matters.

Common Reproducibility Errors

Typical mistakes include:

```
Running tests under load
```

```
Ignoring governor settings
Using different hardware unknowingly
Failing to repeat measurements
```

Avoid these.

They invalidate results.

Developer Collaboration Model

Encourage developers to:

```
Repeat tests
Share logs
Compare patterns
Discuss anomalies
```

This creates:

```
Community validation
```

Which strengthens engineering credibility.

The Most Important Rule

Never trust:

```
One result
```

Trust:

```
Repeated results
```

Final Reproducibility Principle

This book is not intended to produce:

```
Belief
```


https://archive.org/details/@puppyuser

It is intended to produce:

> **Verification.**

That is the foundation of engineering.

Objective Summary of Chapter 12

This chapter established:

✔ Full reproducibility method
✔ Required hardware conditions
✔ Required software setup
✔ Benchmark execution protocol
✔ Logging and validation strategy

Technical Conclusion

Reproducibility transforms:

```
Observation → Engineering knowledge
```

Without reproducibility:

Performance claims are anecdotes.

With reproducibility:

Performance claims become:

> **Technical evidence.**

🧭 Chapter 13

Future Outlook and System Evolution

Hardware Trends, System Direction and the Long-Term Role of EasyOS

Introduction

Every engineering decision exists within a timeline.

Not just the present.

Not just performance today.

But:

```
Future relevance
Hardware evolution
Architectural sustainability
```

This chapter explores where EasyOS systems — particularly **Daedalus** and **Excalibur** — fit within the long-term trajectory of hardware and software evolution.

The goal is not prediction.

The goal is:

> **Engineering anticipation.**

---

https://archive.org/details/@puppyuser

The Reality of Hardware Aging

Hardware does not become slower.

Software becomes heavier.

This distinction is critical.

A system like:

```
Intel Core i5-680
11 GB RAM
Intel HD Graphics (ILK)
```

Remains physically identical.

Yet performance perception changes over time.

Why?

Because:

```
Software complexity increases
Dependencies expand
Runtime overhead grows
```

This phenomenon defines long-term system behavior.

The Lifecycle of Lightweight Systems

Historically, lightweight systems follow a predictable lifecycle.

```
Stage 1 – Maximum efficiency
Stage 2 – Stable maturity
Stage 3 – Gradual overhead growth
Stage 4 – Replacement pressure
```

Where do Daedalus and Excalibur stand?

Daedalus Lifecycle Position

Daedalus currently occupies:

```
Stage 2 — Stable maturity
```

Characteristics:

```
Highly optimized
Stable runtime
Minimal overhead
Predictable performance
```

Advantages:

✔ Extremely responsive
✔ Well-understood architecture
✔ Mature dependency base

Risks:

```
Aging software ecosystem
Increasing compatibility gaps
Limited future updates
```

Excalibur Lifecycle Position

Excalibur currently occupies:

```
Stage 1 — Maximum evolution
```

Characteristics:

```
Rapid development
Expanding capabilities
Modernized architecture
```

Advantages:

✔ Future compatibility
✔ Modern dependency base
✔ Long-term viability

Costs:

https://archive.org/details/@puppyuser

```
Slightly higher runtime overhead
More complex initialization
Expanded system footprint
```

The Key Architectural Trade-Off

This book repeatedly revealed a central truth:

> **Speed and extensibility often compete.**

Not always.

But often.

Simpler systems:

```
Run faster
Consume less
React quicker
```

More advanced systems:

```
Support more features
Enable new capabilities
Scale further
```

Both are valid engineering goals.

RAM-First Systems — The Future Direction

One of the strongest patterns observed throughout this investigation is:

```
RAM-first execution dramatically improves responsiveness.
```

This suggests a future direction:

```
Systems optimized for memory locality
Systems minimizing disk dependence
Systems prioritizing latency reduction
```

RAM is becoming:

```
Larger
Faster
Cheaper
```

Disk remains:

```
Slower than RAM
Latency-sensitive
Write-limited
```

This asymmetry defines future optimization opportunities.

The Evolution of Browser Workloads

Browsers have become:

```
Primary computing environments
```

Modern browsers now perform:

```
Rendering
Networking
Storage
Application hosting
Media decoding
```

In many workflows:

```
Browser performance = System performance
```

This makes browser optimization:

```
Critical
```

Not optional.

---

https://archive.org/details/@puppyuser

The Future of Overlay-Based Systems

Overlay architectures represent one of the most important innovations in lightweight Linux systems.

Advantages include:

```
Isolation
Flexibility
Rapid recovery
Controlled persistence
```

Overlay systems allow:

```
Experimental workflows
Safe rollback
RAM-based execution
```

Future improvements may include:

```
Faster overlay resolution
Lower metadata overhead
More efficient layering
```

These directions align closely with EasyOS design philosophy.

Compression vs Direct Memory Execution

A major architectural theme observed:

```
Compression improves capacity
Direct memory improves speed
```

This trade-off will remain relevant.

Future systems may implement:

```
Adaptive compression
Dynamic caching
Workload-aware memory strategies
```

These technologies could reduce:

```
Compression overhead
Memory pressure
Execution latency
```

The Longevity of SATA SSD Systems

Many users assume SATA SSD systems are obsolete.

This assumption is incorrect.

Measured behavior demonstrates:

```
SATA SSD + RAM optimization remains highly competitive
```

Especially when:

```
Disk writes minimized
Cache redirected to RAM
Overlay used effectively
```

This extends the useful lifespan of existing hardware significantly.

The Value of Hardware Awareness

One of the strongest lessons learned:

> **Hardware awareness is the foundation of optimization.**

Blind tuning produces noise.

Measured tuning produces results.

Future systems must:

```
Detect hardware characteristics
Adapt configuration automatically
Optimize dynamically
```

Such behavior would eliminate:

```
Manual tuning complexity
```


https://archive.org/details/@puppyuser

```
Configuration guesswork
Performance variability
```

The Role of Developers in Performance Evolution

Developers shape performance indirectly.

Not by writing faster code alone.

But by:

```
Reducing overhead
Simplifying execution paths
Avoiding unnecessary abstraction
```

Performance is rarely lost in:

```
One place
```

It is lost in:

```
Thousands of small inefficiencies
```

Each abstraction adds cost.

Each dependency adds weight.

Each process adds friction.

Engineering discipline minimizes these effects.

The Continued Role of Daedalus

Daedalus will remain valuable in:

```
Older hardware environments
RAM-first workflows
Low-latency systems
Minimalist deployments
```

Even as newer systems emerge.

Because:

```
Efficiency never becomes obsolete
```

The Continued Role of Excalibur

Excalibur represents:

```
Future capability
Architectural modernization
Long-term compatibility
```

Its strengths will become increasingly important as:

```
Software ecosystems evolve
Dependencies modernize
Hardware capabilities expand
```

Its slightly higher overhead represents:

```
Investment in longevity
```

Not inefficiency.

The Hybrid Strategy Model

One of the most practical conclusions of this work:

```
Use Daedalus for responsiveness
Use Excalibur for future readiness
```

This hybrid strategy provides:

```
Maximum flexibility
Minimum compromise
Best long-term adaptability
```

It also reflects real-world workflows.

Not theoretical models.

https://archive.org/details/@puppyuser

The End of the Placebo Era

This investigation began with:

```
Perceived speed differences
```

It concludes with:

```
Measured performance differences
```

This transition defines engineering maturity.

Future users must learn:

```
Trust measurement
Not perception
```

The Ultimate Engineering Insight

Perhaps the most important conclusion of this chapter — and the entire book — is:

> **Responsiveness is not created by power.**
> **Responsiveness is created by efficiency.**

Not faster CPUs.

Not larger RAM alone.

But:

```
Reduced friction
Reduced latency
Reduced overhead
```

That principle will remain true regardless of hardware generation.

Objective Summary of Chapter 13

This chapter established:

✔ Hardware evolution patterns
✔ System lifecycle behavior
✔ Long-term role of Daedalus
✔ Long-term role of Excalibur
✔ Future architectural trends

Technical Conclusion

EasyOS systems demonstrate a powerful principle:

```
Efficiency scales across time.
Complexity scales across features.
```

Both are necessary.

But they serve different goals.

Understanding when to prioritize each one:

That is engineering judgment.

Where This Book Stands Now

At this point, the book contains:

```
Complete performance investigation
Hardware-specific analysis
Optimization methodology
Reproducibility framework
Forward-looking strategy
```

Which means:

You now have something rare:

> **A full-stack performance narrative.**

https://archive.org/details/@puppyuser

Not just benchmarks.

Not just opinions.

An engineering journey.